



ALMA MATER STUDIORUM  
UNIVERSITÀ DI BOLOGNA

## Laboratorio di Sicurezza Informatica

# Richiami essenziali di utilizzo della riga di comando Linux

**Marco Prandini**

Dipartimento di Informatica – Scienza e Ingegneria

# Riga di comando

- Lo strumento più potente, flessibile e soprattutto più standard con cui amministrare un sistema Unix è l'interfaccia testuale a **riga di comando**.
- Componente essenziale è la **shell** o interprete dei comandi, che permette di svolgere compiti quali job control, mette a disposizione variabili e strutture di controllo per scrivere semplici programmi, e permette di invocare gli eseguibili installati nel sistema.
  - Possiamo definire shell qualsiasi interfaccia tra utente e OS
- Una delle shell più usate è **bash**, (Bourne-again shell) un'evoluzione della classica Bourne Shell di Unix.
- Una volta inserite le credenziali, il sistema presenta un *prompt* che indica la disponibilità dell'interprete ad accettare comandi

# Chi sono, chi c'è

- Poiché è del tutto comune disporre di differenti identificativi utente con cui lavorare, è utile disporre di un comando per sapere quale e' l'identificativo con il quale si sta operando:
  - **whoami** indica il proprio username
  - **id** dà informazioni sull'identità e sul gruppo di appartenenza
  - **who** indica chi e' attualmente collegato alla macchina

# Shutdown

- Anche lo spegnimento di una macchina Linux (e unix in generale) richiede alcune operazioni.
- Si può abbandonare semplicemente la sessione di lavoro con `exit` (dalla shell di login) o con `logout`. Il sistema rimane comunque attivo e pronto ad altri accessi.
- Lo spegnimento richiede invece, di norma, l'accesso di root.
  - Sempre parlando di terminale. I sistemi grafici lo consentono all'utente.
  - Alcune installazioni permettono a chiunque si trovi in possesso della console (tastiera) di eseguire lo spegnimento (shutdown) premendo CTRL-ALT-CANC.
- root può invocare direttamente lo spegnimento, tipicamente con il comando:  

```
shutdown [-h|-r] now
```

  - `-h` indica la richiesta di arrestare il sistema (altrimenti: comando `halt`),
  - `-r` indica la richiesta di riavviare il sistema (altrimenti: comando `reboot`).
  - `now` indica quando eseguire l'operazione. E' possibile, ad esempio, lasciare agli utenti collegati alcuni minuti per terminare il proprio lavoro.

# Esecuzione dei comandi

La shell indica il proprio stato di 'pronto' con una stringa di caratteri visualizzati nella parte iniziale della prima linea vuota. Questa stringa è detta 'prompt'.

I caratteri digitati dall'utente dopo il prompt e terminati dal fine linea (ritorno a capo) costituiscono la command line.

Questa, tipicamente, contiene comandi e argomenti.  
I comandi digitati sulla command line possono essere:

- keyword
- built-in
- comandi esterni
- alias
- funzioni

# Alias

- La shell mette a disposizione alcuni sistemi per memorizzare linee di comando complesse in comandi più semplici da invocare. Uno di questi è l'**alias** che permette di attribuire un nome ad una command line:

```
alias miols='ls -l'
```

- Le associazioni definite con alias vanno perse al termine della sessione, vedremo come renderle persistenti.

# Documentazione dei comandi (e non solo)

- **man pages** – ogni applicazione installa “pagine di manuale” relative al suo utilizzo e configurazione.
  - Le man pages si leggono con il comando *man*
  - I builtin, non essendo programmi installati indipendentemente, non hanno man page.
    - Un sommario del loro funzionamento può essere visualizzato con **help <builtin>**
    - Inoltre, naturalmente, sono documentati nella man page `bash(1)`
- **info files** – a metà strada tra la man page e l'ipertesto, si leggono con il comando *info*, che invoca l'editor emacs appositamente esteso per gestire questi file
- **HOWTO** – documenti specifici per la risoluzione dei più svariati problemi pratici, sono raccolti in un pacchetto e vengono tutti installati in `/usr/[share/]doc/HOWTO`  
Inoltre, sotto `/usr/doc/HOWTO/translations/it` si possono trovare la maggior parte degli HOWTO tradotti in italiano.
- **on-line** – troppe fonti per citarle... un punto di partenza può essere <http://tldp.org/> The Linux Documentation Project

# Categorie di man pages

Una installazione standard di unix mette a disposizione innumerevoli **pagine di manuale** raggruppate in sezioni:

- (1) User commands
- (2) Chiamate al sistema operativo
- (3) Funzioni di libreria, ed in particolare
- (4) File speciali (/dev/\*)
- (5) Formati dei file, dei protocolli, e delle relative strutture C
- (6) Giochi
- (7) Varie: macro, header, filesystem, concetti generali
- (8) Comandi di amministrazione riservati a *root*
- (n) Comandi predefiniti del linguaggio Tcl/Tk



# Accesso alle man pages

- L'accesso alle pagine si ottiene con il comando
- **man** <nome della pagina>
- Spesso il nome della pagina coincide con il comando o il nome del file di configurazione che essa documenta.
- Alcune opzioni utili sono qui riassunte:
  - **man -a** <comando> cercherà in tutte le sezioni
  - **man <sez.>** <comando> cercherà nella sezione specificata
  - **man -k** <keyword> cercherà tutte le pagine attinenti alla parola chiave specificata

Per avere altre informazioni sul comando man è ovviamente sufficiente usare man man.

# Gli argomenti

- Ogni comando, sia built-in che esterno, può accedere ai caratteri che seguono la propria invocazione sulla command line.
  - La shell inserisce in memoria l'ARGV prima di generare il processo
- I gruppi di caratteri, **separati da spazi**, rappresentano gli **argomenti**, cioè i dati su cui si vuole che il comando operi.
- Un argomento che inizia con il carattere '-' è chiamato solitamente *opzione*
  - normalmente non è un vero e proprio dato da elaborare
  - è un modo di specificare una variante al comportamento del comando
  - più opzioni possono essere solitamente raggruppate in un'unica stringa.

# Esempi di argomenti e opzioni

ls /home

**arg #1="/home"** indica la directory di cui elencare il contenuto

ls -l /home

**arg #1=opzione l** indica che si desidera un listato in forma "lunga"

ls -l -a /home/alex

**arg #1 e #2 = opzioni l** (lunga) ed **a** (all)

ls -la /home/alex

**arg #1 = opzioni concatenate l+a**  
(identico a prima)

# Digitare interattivamente la command line

- Iniziando a scrivere un comando (come primo elemento) o un nome di file (come argomento), e battendo TAB, bash completa automaticamente la stringa se non ci sono ambiguità, o suggerisce come completarla correttamente.

Es.:

al prompt digito **pass**<TAB> → compare **passwd** seguito da spazio ad indicare che passwd è l'unico completamento possibile

- Se ci sono ambiguità, una seconda pressione di <TAB> mostra i completamenti possibili.

Es.

- digito **ls /etc/pam**<TAB> → compare **/etc/pam.**
- digito <TAB> → vengono elencati **pam.conf** e **pam.d/**
- aggiungo "**c**" e digito <TAB> → compare **/etc/pam.conf**

# Richiamare comandi passati

- **history** mostra l'elenco di tutti i comandi eseguiti in un terminale. Per richiamarli sulla command line, basta usare la freccia-su, appariranno (editabili) dal più recente al più vecchio
- La history è anche ricercabile interattivamente: al prompt basta digitare **CTRL-r** per far apparire un prompt (reverse-i-search); digitando una stringa, verrà mostrato il comando più recente che la contiene.
- Per navigare verso comandi impartiti precedentemente basta digitare nuovamente **CTRL-r**.
- Individuato il comando desiderato, si può lanciare direttamente premendo invio, o renderlo editabile sulla command line con freccia-destra o freccia-sinistra

# Documentare le attività svolte sulla shell

- Il comando **script** permette di catturare in un file una sessione di lavoro al terminale, esattamente come compare a video, quindi sia i comandi impartiti che il loro risultato.
- Per terminare l'attività, digitare **exit** o premere **CTRL-d**

# Comandi utili per lavorare coi file

- Elenco e navigazione
- Analisi dei metadati
- Trasferimento dati
- Ricerca nel filesystem
- Archiviazione e compressione

# Navigazione

- Il filesystem Linux è un unico albero, anche in presenza di più dischi fisici.
- La radice è rappresentata dalla “barra dritta” /
  - I path completi quindi si presentano, ad esempio, così: `/usr/bin/passwd`
- Quando si opera sulla riga di comando, la shell usa il concetto di directory di lavoro (`pwd` – present working directory),
  - in ogni momento i path relativi fanno riferimento alla `pwd` in cui ci si trova
  - un processo continua a occupare la `pwd` in cui era l'utente che lo ha lanciato
- `pwd` mostra la directory corrente di lavoro
- `cd` permette di spostarsi a un'altra directory
  - esplicitamente nominata come parametro, oppure
  - la home dell'utente, se invocato senza parametri, oppure
  - la directory in cui ci si trovava prima dell'ultimo `cd`, se invocato con parametro `-`
- ricordiamo che in ogni directory `D` sono sempre presenti due sottodirectory
  - `.` che coincide con la directory `D` stessa
  - `..` che coincide con la directory superiore (in cui `D` è contenuta)



# Le marcature temporali (timestamp)

- Ogni file ha tre (o quattro) timestamp distinti
  - mtime      modification time      istante dell'ultima **modifica** del **contenuto**
  - atime      access time      istante dell'ultimo **accesso** al contenuto
  - ctime      change time      istante dell'ultima **modifica ai metadati**
  - wtime      birth time      istante della **creazione** del file, **se supportato dal filesystem**
- Queste informazioni vengono gestite automaticamente dal filesystem, ma possono essere cambiate a mano col comando **touch**
- Tutti i metadati possono essere estratti e visualizzati in un formato arbitrario col comando **stat**

```
stat --format='%U %a %z' /etc/passwd
```

```
root 644 2021-03-15 08:33:06.381876582 +0100
```

(U=utente proprietario, a=permessi, z=ctime)

# Opzioni principali di **ls**

- l** abbina al nome le informazioni associate al file
- a** non nasconde i nomi dei file che iniziano con .
  - per convenzione i file di configurazione iniziano con un punto, non essendo interessanti per l'utente non sono mostrati di default da **ls**
- A** come -a ma esclude i file particolari . e ..
- F** pospone il carattere \* agli eseguibili e / ai direttori
- d** lista il nome delle directory senza listarne il contenuto
  - il comportamento di default di ls quando riceve come parametro una directory è di elencarne il contenuto, cosa spesso indesiderabile quando nomi di file e directory vengono espansi dalla shell a partire da wildcard
- R** percorre ricorsivamente la gerarchia
- i** indica gli i-number dei file oltre al loro nome
- r** inverte l'ordine dell'elenco
- t** lista i file in ordine di data/ora di modifica (dal più recente)

# Identificazione del contenuto di file

- In Linux, le estensioni dei nomi hanno come unico utilizzo quello di renderli più leggibili all'utente
- Si può ottenere manualmente l'identificazione con **file**
  - test 1: usa stat per capire se il file è vuoto o speciale
  - test 2: usa il database dei **magic number** per identificare il file
  - test 3: usa metodi empirici per capire se è un file di testo, e in tal caso quale sia la lingua naturale o linguaggio di programmazione

# Creazione e rimozione di file

- **rm** cancella un file o, meglio, rimuove il link
- **cp** copia un file o piu' file in una directory
- **mv** sposta un file o piu' file in una directory
- **ln** crea un link ad un file
  - hardlink di default, solo all'interno dello stesso FS e non verso directory
  - symlink con l'opzione **-s**, nessuna limitazione
- **mkdir** crea una directory
- **rmdir** cancella una directory
  - deve essere vuota
  - **rm -r** cancella ricorsivamente

# Trasferimento di dati

- I comandi più ovvi non sono pratici per trasferire dati da/verso file speciali
  - **cat** e ridirezioni sono utilizzabili in modo “tutto o niente”
  - **cp** non è utilizzabile
- **dd** permette di leggere byte da qualsiasi file (**if=<NOME>**) e scrivere su qualsiasi file (**of=<NOME>**)
  - se **NOME** = **-** si intende STDIN (per **if**) o STDOUT (per **of**)specificando
  - da che punto iniziare a leggere **skip=<N>**
  - in che punto iniziare a scrivere **seek=<N>**
  - quanti dati trasferire **count=<N>**
  - con che dimensione di blocco operare **bs=<N>**inoltre può eseguire trasformazioni di formato e tracciare il progresso del trasferimento

# Ricerca nel filesystem con find

## ■ find ricerca in tempo reale

- quindi esplorando il filesystem → attenzione al carico indotto!

i file che soddisfano una combinazione di criteri, ad esempio:

- nome che contenga una espressione data
- timestamp entro un periodo specificato
- dimensione compresa tra un minimo e un massimo
- tipo specifico (file, dir, link simbolici, ...)
- di proprietà di un utente o di un gruppo specificati (o “orfani”)
- permessi di accesso specificati

e molti altri

## ■ Esempio:

- ricercare sotto `/usr/src` tutti i file che finiscono per `.c`, hanno dimensione **maggiore di 100K**, ed elencarli sullo standard output:

```
find /usr/src -name '*.c' -size +100k -print
```

# Esecuzione di operazioni sui file trovati

- Una delle opzioni più potenti di `find` permette, per ciascun oggetto individuato secondo i criteri impostati, di invocare l'esecuzione di un comando:
- Es. mostra il contenuto dei file trovati
  - `find /usr/src -name '*.c' -size +100k -exec cat {} \;`
    - il comando che segue `-exec` viene lanciato per ogni file trovato
    - la sequenza `{}` viene sostituita di volta in volta con il nome del file
    - `\;` è necessario per indicare a `find` la fine del comando da eseguire
- Es. elenca solo i file regolari “orfani” modificati meno di due giorni (2\*24 ore) fa che contengono TXT

```
find / -type f -nouser -mtime -2 -exec grep -l TXT {} \;
```

# Ricerca di file con locate

- **locate** effettua la ricerca su di un database indicizzato
  - Il database deve essere aggiornato periodicamente con l'utility `updatedb`
- **Vantaggi su find**
  - Carico sul sistema ridotto a una singola esplorazione per ogni periodo, indipendentemente dal numero di query successive
  - Esplorazione pianificabile nei momenti di basso carico
  - Risposta pressoché istantanea
- **Svantaggi rispetto a find**
  - Unico criterio di ricerca: pattern nel nome
  - Risposte potenzialmente obsolete
    - file creati dopo l'esplorazione non vengono riportati
    - file cancellati dopo l'esplorazione sembrano ancora esistere



# Archiviazione di file

- Per poter agevolmente memorizzare e trasferire una molteplicità di file, eventualmente senza perdere le proprietà associate a ciascuno (ownership, permessi, timestamps...) è comune avvalersi di **tar**. La sintassi prevede che debba essere specificato esattamente uno dei seguenti comandi:

<b>-A</b>	concatena più archivi
<b>-c</b>	crea un nuovo archivio
<b>-d</b>	trova le differenze tra archivio e filesystem
<b>-r</b>	aggiunge file ad un archivio
<b>-t</b>	elenca il contenuto di un archivio
<b>-u</b>	aggiorna file in un archivio
<b>-x</b>	estrae file da un archivio
<b>--delete</b>	cancella file da un archivio

# Archiviazione di file

- Le origini di **tar** risalgono ai tempi dei nastri magnetici (il nome è acronimo di Tape ARchiver) quindi di default assume che l'archivio sia su **/dev/tape**.
- L'opzione **-f <FILENAME>** viene quindi sempre usata per specificare un file di archiviazione.
  - Dove sensato, **FILENAME** può essere **-** in per indicare
    - lo standard input da cui leggere un archivio con **d, t, x**
    - lo standard output su cui scrivere l'archivio con **c**
- Altre opzioni comunemente usate sono:
  - p** (preserve) conserva tutte le informazioni di protezione
    - (funziona pienamente solo per root, un utente standard quando ricrea i file estraendoli da un archivio è forzato a dargli la sua ownership)
  - v** stampa i dettagli durante l'esecuzione
  - T <ELENCO>** prende i nomi dei file da archiviare da **ELENCO** invece che come parametri sulla riga di comando
  - C <DIR>** svolge tutte le operazioni come dopo **cd DIR**

# Archiviazione di file

- Esempi (si noti che il trattino per indicare le opzioni può essere omesso fintanto che non è necessario utilizzare più di un'opzione che richiede parametri)

- creazione

```
tar cvpf users.tar /home/*
```

- la barra iniziale verrà rimossa in modo da rendere relativi tutti i path

- estrazione

```
tar -C /newdisk -xvpf users.tar
```

- poiché i path nell'archivio sono relativi, la directory home viene ricreata dentro /newdisk e tutta la gerarchia sottostante viene ricostruita

- pipeline

```
tar cvpf - /home/* | tar -C /newdisk -xvpf -
```

# Compressione di file

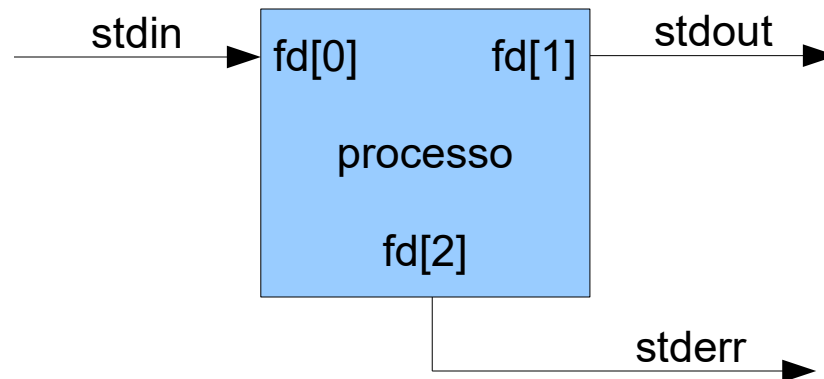
- tar non comprime
- esistono moltissimi formati di compressione
  - [https://linuxhint.com/top\\_10\\_file\\_compression\\_utilities\\_on\\_linux/](https://linuxhint.com/top_10_file_compression_utilities_on_linux/)
  - [https://en.wikipedia.org/wiki/List\\_of\\_archive\\_formats](https://en.wikipedia.org/wiki/List_of_archive_formats)
- I più comuni nei sistemi Linux sono
  - estensione **.gz**                      comando base: **gzip**
  - estensione **.bz2**                      comando base: **bzip2**
  - estensione **.xz**                      comando base: **xz**
- Il comando base prende come argomento un file e lo comprime aggiungendo l'estensione
  - con l'opzione **-d** decomprime ricreando il file e rimuovendo l'estensione
  - con l'opzione **-c** riversa il risultato su STDOUT invece che su file
    - filtro!
    - es: **tar cf - \* | xz -c > archive.tar.xz**

# Compressione di file - scorciatoie

- Esiste tipicamente un comando di decompressione equivalente al comando base invocato con -d
  - es. **gunzip**, **bunzip2**, **unxz**
- Esistono alias per le combinazioni più comuni di filtro di decompressione e comandi di trattamento testo
  - **zcat file.gz == gzip -dc file.gz**
  - **zegrep <REGEX> file.gz == gzip -dc file.gz | egrep <REGEX>**
  - (idem per i decompressori **bz\***, **xz\***, e per i comandi **\*diff**, **\*less**, **\*cmp**)
- **tar** in particolare supporta opzioni per invocare direttamente la (de)compressione di un archivio
  - z**            usa **gzip**            estensione **.tar.gz** o **.tgz**
  - j**            usa **bzip2**            estensione **.tar.bz2** o **.tbz2**
  - J**            usa **xz**            estensione **.tar.xz** o **.txz**
  - esempio precedente == **tar cJf archive.tar.xz \***

# Comporre comandi e file

- Per convenzione quindi tutti i comandi \*nix che operano su stream di testo (filtri) sono progettati per disporre di tre stream con cui comunicare con il resto del sistema:
  - standard input in ingresso (file descriptor 0)
  - standard output in uscita (file descriptor 1)
  - standard error in uscita (file descriptor 2)



# Ridirezione da/verso file

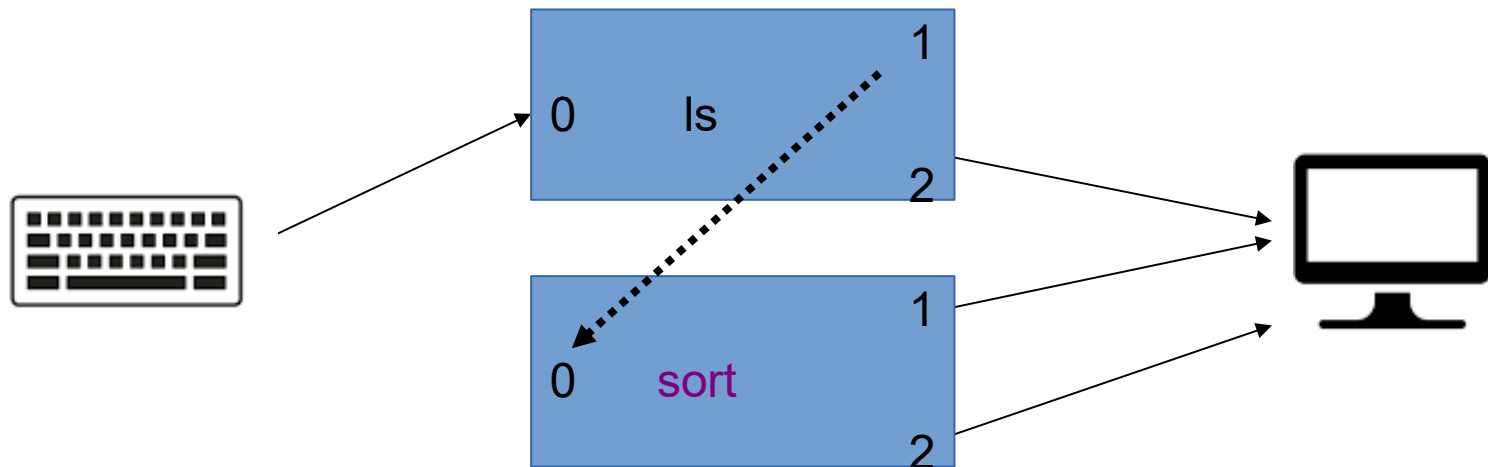
- Bash, nell'interpretare la riga di comando, può **disconnettere gli stream predefiniti dal terminale** (chiudendoli nel processo figlio dopo la fork) e far trovare gli stessi file descriptor aperti su di un file diverso (aprendolo prima della exec)
- Ridirezione dello stdout: **>** e **>>**
  - `ls > miofile` scrive lo stdout di ls nel file miofile (troncandolo)
  - `ls >> miofile` scrive lo stdout di ls nel file miofile (in append)
  - se miofile non esiste viene creato
- Ridirezione dello stderr: **2>** e **2>>**
  - come sopra ma ridirige lo stderr
- Confluenza degli stream
  - `ls > miofile 2>&1` ridirige lo stderr dentro stdout e poi stdout su file  
**l'ordine è importante!**
- Ridirezione dello stdin **<**
  - `sort < miofile` riversa il contenuto di miofile su stdin di sort

# bash pipe

- Cosa succede quando si esegue

```
ls | sort
```

- Bash prepara il terreno perché ciò che **ls** produce su **stdout** venga riportato su **stdin** di **sort**





# Filtri utili

- Il meccanismo di ridirezione è utilizzato da un set di comandi pensati esattamente per elaborare stream di testo ricevuti via stdin, che producono risultati su stdout, i *filtri*
- Vedremo alcuni dei più comuni
- Molti di questi permettono comunque di operare direttamente anche su file specificati come parametro, da cui prelevano i dati da elaborare in alternativa alla lettura da stdin

# Concatenazione di file – cat / tac

- Il più semplice dei filtri: invocato senza parametri copia stdin su stdout
- invocato con uno o più file come parametri, ne produce in sequenza il contenuto su stdout

```
cat file1 file2
```

qualche minima opzione utile: numerare le righe, evidenziare tab e fine linea, ecc.

```
man cat
```

- **tac** riproduce le righe in ingresso (stdin o file) su stdout in ordine inverso, dall'ultima alla prima

# Impaginazione - less

- Non è realmente un filtro perché l'output è destinato al terminale, ma è tra i comandi più utili per l'uso interattivo
- Posto al termine di una pipeline, intercetta l'output e lo mostra riempiendo lo spazio disponibile sul terminale, permettendone la navigazione per mezzo di vari comandi (tipicamente la pressione di un singolo tasto)
- Comandi principali (ce ne sono altre decine - **man less**):
  - **h** help (dei comandi disponibili)
  - **frecce/pag-su/giu** movimento
  - **F** Follow; scorre fino al termine dell'input e resta in attesa che compaiano nuove righe da mostrare
  - **<N>g** si porta alla riga numero **<N>** (default: 1)
  - **G** si porta al termine del file
  - **/**<pattern>**** cerca la riga successiva al cursore contenente **<pattern>**
  - **?<pattern>**  
**<pattern>** cerca la riga precedente il cursore contenente **<pattern>**
  - **n** ripete la ricerca fatta in precedenza
  - **N** ripete la ricerca fatta in precedenza, ma nel verso opposto
  - **q** esce da less

# Il comando rev

- rev è un filtro che permette di invertire l'ordine dei caratteri di ogni linea dello stream in input verso lo stream di output.
- L'utilità del comando è solitamente quella di accompagnare cut nella estrazione di campi la cui posizione sia nota relativamente al fine linea:

```
cat /etc/passwd | rev | cut -f1 -d: -s | rev
```

elabora il contenuto del file /etc/passwd nel seguente modo:

- inverte ogni linea
- prende il primo campo → l'ultimo campo dell'originale
- inverte ogni linea (ripristina il campo selezionato)

# I comandi head e tail

- **head** è un filtro che permette di estrarre la parte iniziale di un file
  - default: prime 10 righe
  - **-c NUM** produce i primi **NUM** caratteri
    - usando **-NUM** produce tutto il file eccetto gli ultimi **NUM** caratteri
  - **-n NUM** produce le prime **NUM** righe
    - usando **-NUM** produce tutto il file eccetto le ultime **NUM** righe
  
- **tail** è un filtro che permette di estrarre la parte finale di un file
  - default: ultime 10 righe
  - **-c NUM** produce gli ultimi **NUM** caratteri
    - usando **+NUM** produce tutto il file a partire dal carattere **NUM**
  - **-n NUM** produce le ultime **NUM** righe
    - usando **+NUM** produce tutto il file a partire dalla riga **NUM**

# Opzioni particolari di tail

- Un'opzione particolarmente importante di tail è **-f** con cui, dopo aver mostrato le ultime righe di un file, lo si mantiene aperto e si visualizzano in tempo reale eventuali nuove righe che vi vengano appese da altri processi
- **tail -f** quindi può essere usato per monitorare l'output che un processo sta scrivendo su di un file
  - per far questo ignora il raggiungimento di EOF... come termina?
    - con **-f** si può usare **--pid=PID** per fare in modo che alla terminazione del processo PID termini anche tail
  - e se il produttore è in ritardo, e non genera il file in tempo per l'avvio di tail?
    - con **--retry**, tail continuerà a provare finché non riuscirà ad aprire il file
  - **-F == -f --retry**

# Estrazione dei caratteri di una riga – cut

- Il comando cut permette di tagliare parti di righe.
- Il modo di operazione più semplice è attivato dall'opzione **-c**

```
cut -cELENCO_POSIZIONI_CARATTERI [input_file]
```

“ritaglia” le righe, producendo per ogni riga in ingresso una riga in uscita composta dai soli caratteri elencati.

Esempi:

<code>cut -c15</code>	restituisce solo il 15° carattere
<code>cut -c8-30</code>	restituisce i caratteri dall'8° al 30°
<code>cut -c-30</code>	restituisce i caratteri fino al trentesimo
<code>cut -c8-</code>	restituisce i caratteri dall'ottavo in poi

# Estrazione dei campi – cut

- Su file organizzati a 'record' (uno per riga) per i quali ogni record rappresenta una lista di 'campi' opportunamente delimitati, le opzioni **-d** e **-f** di cut permettono di estrarre uno o più campi di ciascun record

**cut -dCARATTERE\_DELIMITATORE -fELENCO\_CAMPI**

Es. se ci interessa estrarre solo il campo username dal file passwd:

**cat /etc/passwd | cut -f1 -d: -s**

Es. se nel campo note metto 'Nome Cognome' e voglio l'iniziale dei cognomi degli utenti:

- **cat /etc/passwd | cut -f5 -d: -s | cut -f2 -d' ' | cut -c1**

-s evita che vengano prodotte in output le righe che non contengono il delimitatore (che altrimenti sono riprodotte per intero)



# I comandi sort e uniq

- Per ordinare le linee di uno stream o per individuare le righe duplicate o uniche, unix mette a disposizione i filtri **sort** e **uniq**.

- **sort** ordina in modo lessicale

- ordine dei caratteri =
  - se **LC\_ALL=C** → valore dei byte che li codificano
  - diversamente → ordine stabilito dal *locale* scelto
- opzioni che controllano il comportamento globale
  - u** elimina le entry multiple (equivale a **sort | uniq**)
  - r** reverse (ordinamento decrescente)
  - R** random (permutazione casuale delle righe)
  - m** merge di file già ordinato
  - c** controlla se il file è già ordinato

- **uniq** elimina i duplicati consecutivi

- c** indica anche il numero di righe compattate in una
- d** mostra solo le entry non singole

# sort - opzioni avanzate di ordinamento

- Oltre all'ordinamento di default, sort è in grado di confrontare le righe sulla base di altri criteri
    - b ignora gli spazi a inizio riga
    - d considera solo i caratteri alfanumerici e gli spazi
    - f ignora la differenza minuscole / maiuscole
    - n interpreta le stringhe di numeri per il valore numerico
    - h interpreta i numeri "leggibili" come 2K, 1G, ecc.
  - inoltre, può essere istruito a cercare le chiavi di ordinamento in posizioni specifiche della riga, invece che considerarla per intero
    - tSEP imposta SEP come separatore tra campi (default: spazi)
    - kKEY chiave di ordinamento – se usato più volte, ordina per la prima chiave, a parità di questa per la seconda, ...
- KEY è nella forma (semplificata) **F[.C][,F[.C]][OPTS]**
- **F** = numero di campo
  - **C** = posizione (in caratteri) nel campo
  - **OPTS** = una delle opzioni di ordinamento **[bdfgiMhnRrV]**
- Es: **sort -t. -k 1,1n -k 2,2n -k 3,3n -k 4,4n**  
ordina un elenco di IP address (byte1.byte2.byte3.byte4)

# Il comando **wc**

- **wc** (word count) è un filtro di conteggio
  - c** conta i caratteri
  - l** conta le linee
  - w** conta le parole (stringhe separate da spazi)

# Ricerca di parti in un testo: Grep

- Esamina le righe del testo in ingresso (su standard input o specificato come elenco di file sulla riga di comando)
- Riproduce in uscita quelle che contengono un pattern corrispondente a una **espressione regolare** (nel caso più semplice una sottostringa) passata come argomento.

Es. per cercare una stringa di nome "prova" all'interno dell'output di ls:

```
ls | grep prova
```

- Da qui in avanti faremo riferimento alla variante di grep che supporta le espressioni regolari “moderne”: **egrep**

# Espressioni regolari moderne (o estese)

- **egrep** utilizza una sintassi per la specifica dei pattern di ricerca detta espressione regolare (regexp o **RE**). La documentazione è reperibile nella man page **regex(7)**.
- In sintesi
  - **RE** = uno o più **rami** non vuoti separati da |
  - **ramo** = uno o più **pezzi** concatenati tra loro
  - **pezzo** = **atomo** eventualmente seguito da un *moltiplicatore*
  - **atomo** = uno di
    - **(RE)**
    - **[charset]**
    - **^ o \$ o .**
    - backslash sequence
    - Singolo carattere

# Espressioni regolari moderne (o estese)

## ■ Atomi speciali:

- `.` indica UN qualsiasi carattere
- `^` indica l'inizio della linea
- `$` indica la fine della linea

## ■ Backslash sequence:

- `\< – \>` la stringa vuota all'inizio – alla fine di una parola.
- `\b` la stringa vuota al confine di una parola
- `\B` la stringa vuota a condizione che non sia al confine di una parola
- `\w` è sinonimo di “una qualsiasi lettera, numero o \_”
- `\W` è un sinonimo “un qualsiasi carattere non compreso in \w”

## ■ Moltiplicatori:

- `{n,m}` indica da n a m occorrenze dell'atomo che lo precede
- `?` indica zero o una occorrenza dell'atomo che lo precede
- `*` indica zero o più occorrenze dell'atomo che lo precede
- `+` indica una o più occorrenze dell'atomo che lo precede

# Espressioni regolari moderne (o estese)

## ■ Charset – esempi:

`[abc]` indica UN qualsiasi carattere fra a, b o c

`[a-z]` indica UN qualsiasi carattere fra a e z compresi

`[^dc]` indica UN qualsiasi carattere che non sia né d né c

## ■ Charset basati su character class

`[ :NOME_CLASSE : ]`

dove ***NOME\_CLASSE*** deve appartenere all'insieme definito in `wctype(3)` come tipicamente:

<code>alnum</code>	<code>digit</code>	<code>punct</code>	<code>alpha</code>	<code>graph</code>	<code>space</code>
<code>blank</code>	<code>lower</code>	<code>upper</code>	<code>cntrl</code>	<code>print</code>	<code>xdigit</code>

o eventualmente nel *locale* attivo

# grep – regole di matching

## ■ *Greediness*

- Nel caso in cui una RE possa corrispondere a più di una sottostringa di una data stringa, la RE corrisponde a quella che inizia per prima nella stringa
- Se a partire da quel punto, la RE può corrispondere a più di una sottostringa, selezionerà la più lunga.


## ■ Priorità nelle RE multilivello

- Anche le sottoespressioni selezioneranno sempre le sottostringhe più lunghe possibili
  - salvo il vincolo che l'intera corrispondenza sia la più lunga possibile
  - dando la priorità alle sottoespressioni che iniziano prima nella RE su quelle che iniziano dopo

Si noti che le sottoespressioni di livello superiore hanno quindi la priorità sulle loro sottoespressioni di componenti di livello inferiore.



# Espressioni regolari - esempi

- Iniziamo osservando che molti caratteri speciali delle RE sono anche caratteri speciali della shell – quando possibile, a scanso di equivoci, è meglio racchiudere l'intera RE tra apici
- `egrep '^Nel.*vita\.$'` miofile  
ha come output tutte le righe di miofile che iniziano per **Nel** e finiscono per **vita.**  
 notare il quoting del punto nella RE
- `egrep '.es[^es]{3,5}e'` miofile  
ha come output tutte le righe che contengono in qualsiasi posizione la sequenza:
  - 1 carattere qualsiasi
  - **es**
  - una sequenza di 3-5 caratteri potenzialmente diversi uno dall'altro a patto che ognuno sia diverso da **e** ed **s**
  - **e**
- Per più esempi, basta andare online, es.  
<https://www.cyberciti.biz/faq/grep-regular-expressions/>
- Ci sono anche tester online ma attenzione al dialetto delle RE usato!

# Grep – opzioni principali

## ■ Controllo del tipo di matching:

- E** usa le extended RE (come egrep senza parametri)
- F** disattiva le RE e usa il parametro come stringa letterale
- w / -x** fa match solo con RE “whole word” o “whole line”
- i** rende l’espressione insensibile a maiuscole e minuscole

## ■ Controllo dell’input

- r** cerca ricorsivamente in tutti i file di una cartella
- f *FILE*** prende le RE da un **FILE** invece che come parametro

## ■ Controllo dell’output

- o** restituisce solo le sottostringhe che corrispondono alla RE  
**invece** della riga che le contiene, separatamente una per riga di output.
- v** restituisce le linee che non contengono l’espressione
- l** utile passando a grep più file su cui cercare: restituisce solo i nomi dei file in cui l’espressione è stata trovata
- n** restituisce anche il numero della riga contenente l’espressione
- c** restituisce solo il conteggio delle righe che contengono la RE
- line-buffered** disattiva il buffering

# Gestione dei processi

- Qualsiasi processo è originato dall'esecuzione di un programma
  - Il programma è un file, statico, memorizzato su disco
  - Il processo è il programma caricato in memoria + tutti i metadati che consentono al sistema operativo di schedarlo e di tracciare le risorse che usa
- Ai fini pratici, distinguiamo due categorie
  - Processi lanciati automaticamente e “nascosti” sullo sfondo (demoni in background)
  - Processi lanciati esplicitamente dalla riga di comando

# Gestione dei demoni

- Nella maggior parte delle distribuzioni moderne è gestita da systemd

- L'interfaccia verso systemd è il comando systemctl (da usare con privilegi di root)

```
systemctl {start|stop|status|restart|reload} servicename
```

—

- Tipicamente, ogni volta che si cambia la configurazione di un demone, bisogna riavviarlo (reload o restart)

# Gestione dei processi da terminale

## ■ **ps** – process status

- Nella sua semplicità concettuale, supporta un numero strabiliante di opzioni.
- Visualizzare tutti i processi con molti dettagli `ps auxw`
- Poi meglio avere sotto mano un cheat sheet
- <https://www.golinuxcloud.com/ps-command-in-linux/>
- **top** mostra i processi più attivi e altre stats

## ■ Ogni processo ha un identificatore univoco (PID)

## ■ Per inviare un segnale a un processo si può usare

**kill [options] <pid> [...]**

- PID negativi identificano l'intero process group
- l'opzione **-l** / **-L** elenca i segnali supportati

# Invio di segnali con combinazioni di tasti

- Il terminale trasforma la ricezione di alcune combinazioni di tasti in segnali inviati al processo che lo sta occupando:

**Ctrl + Z** → SIGTSTP

**Ctrl + C** → SIGINT

**Ctrl + \** → SIGQUIT

- osservazione a lato: il terminale genera anche altri effetti di controllo non legati ai segnali, come

– **eof** = ^D

– **start** (scrolling) = ^Q

– **stop** (scrolling) = ^S

# sleep

- Il comando **sleep** innesca un timer per far “dormire” il processo
- Il parametro può essere un float
  - di default interpretato in secondi
  - sono supportati i suffissi **m**(inutes) **h**(ours) **d**(ays)
- Interazioni coi segnali – valgono le regole di qualsiasi altro comando lanciato dalla shell
  - sleep è un comando esterno
    - genera un processo figlio
    - **mandare un segnale alla shell che lo ha lanciato non lo tocca**
  - sleep invoca una system call che sospende il processo
    - fino al termine della sleep il processo non rientra in user mode
    - **i segnali sono ricevuti ma non processati**

# Processi in background

- Si può usare un'unica shell per l'esecuzione contemporanea di più comandi che non abbiano necessità di accedere al terminale, lanciandoli in **background** (sullo sfondo).
- Questo si ottiene postponendo il carattere **&** alla command line.
  - La shell risponde comunicando un numero tra parentesi quadre (**job id**) che identifica il job **localmente** a questa shell.
    - per usarlo al posto di un PID, si utilizza **%job\_id**
  - **MOLTO UTILE:** Il PID del processo viene memorizzato nella variabile **\$!**
- Se si lancia una command line senza **&**, e si vuole rimediare, si può dare un segnale di **STOP** con Ctrl+Z.
  - Anche in questo caso si riceve un job id.
  - Con il comando **bg %job\_id**, si invia un segnale CONT che riavvia il processo e contemporaneamente lo si mette in background.



# jobs e foreground

- Un processo in background non riceve più comandi dal terminale, poiché la tastiera torna ad agire sulla shell;
  - continua però a utilizzare il terminale per STDOUT e STDERR
- se è necessario riportare in **foreground** (primo piano) un processo ricollegandolo così al terminale, si usa il comando **fg %job\_id**.
- Il comando **jobs** mostra l'elenco dei job, cioè di tutti i processi avviati dalla shell corrente, indicando il loro stato (attivo o stoppato).
- Per esempi e approfondimenti sulla propagazione di segnali a child process:  
<https://linuxconfig.org/how-to-propagate-a-signal-to-child-processes-from-a-bash-script>

# Brevissima introduzione all'editor VIM

- VIM e' un editor a 'tutto schermo', versione più amichevole dello storico VI.
- Utilizza una interfaccia 'modale'. In altre parole, il programma si puo trovare in uno dei seguenti stati:
  - **COMMAND**
    - Il cursore è posizionato sul testo
    - la tastiera è utilizzabile solo per richiedere l'esecuzione di comandi, e non per introdurre testo.
    - I caratteri digitati non vengono visualizzati.
  - **INPUT**
    - Tutti i caratteri digitati vengono visualizzati ed inseriti nel testo.
  - **DIRECTIVE**
    - Ci si trova posizionati con il cursore nella linea direttive (l'ultima linea del video) e si possono richiedere tutti i comandi per il controllo del file.

# Passaggi di stato

- I passaggi di stato avvengono digitando uno dei seguenti caratteri:

- COMMAND MODE → INPUT MODE      oOiIaACR
- INPUT MODE → COMMAND MODE      <ESC>
- COMMAND MODE → DIRECTIVE MODE      : / ?
- DIRECTIVE MODE → COMMAND MODE      <RET>

# Gestione file

- I comandi per caricare/salvare/uscire sono **DIRECTIVE**
  - **:r** <nome> inserisce il contenuto del file **nome** al punto del cursore
  - **:w** scrive il file corrente
  - **:q** esce (**:q!** se si vuole uscire senza salvare)
  - **ZZ** scrive ed esce

# Comandi di spostamento

- In COMMAND MODE, e' possibile richiedere lo spostamento del cursore con i comandi di movimento:

- **h** 1 spazio a sinistra (come backspace)
- **l** 1 spazio a destra (come space)
- **k** 1 linea sopra (stessa colonna)
- **j** 1 linea sotto (stessa colonna)

(in sostituzione di questi quattro caratteri, si possono anche utilizzare le frecce per spostarsi nelle diverse direzioni)

- Inizio/fine riga

- **^** posizionamento all'inizio della riga
- **\$** posizionamento alla fine della riga

- Inizio/fine file

- **gg** posizionamento sulla prima linea del testo
- **G** posizionamento sull'ultima linea del testo
- **#G** posizionamento sulla linea numero "#"

- Esempio: **3G** **5G** **175G**
- Notate che i comandi non vengono visualizzati!

# Modifica

- In COMMAND MODE è anche possibile apportare modifiche
- cancellazione
  - **x** cancella il carattere su cui si trova il cursore
  - **dd** cancella la linea su cui si trova il cursore
- modifica di singoli elementi carattere (non si passa a INPUT MODE)
  - **rX** rimpiazza il carattere sotto il cursore con **X**
- modifica (provocano l'ingresso in INPUT MODE):
  - **i** entra in modalità 'inserimento' nella posizione del cursore
  - **I** entra in modalità 'inserimento' a inizio riga
  - **a** entra in modalità 'append' nella posizione del cursore
  - **A** entra in modalità 'append' a fine riga
  - **R** entra in modalità 'replace' (sovrascrittura)
  - **cw** modalità 'change word', elimina la parola che inizia sotto il cursore ed edita
  - **C** entra in modalità 'change' (elimina fino a fine riga)
  - **o** inserisce una linea vuota sotto al cursore
  - **O** inserisce una linea vuota sopra al cursore

# Ricerca e sostituzione

- Digitando la barra, si entra in DIRECTIVE MODE per cercare stringhe
  - es. `/ciao<RET>` posiziona il cursore sulla prima occorrenza successiva della stringa `ciao` (se esiste)
  - Digitando `n` si passa alla successiva (next)
  - Digitando `N` si passa alla precedente
- Se al posto di `/` si usa `?`, la direzione di ricerca è verso l'inizio (e i significati di `n` e `N` si adeguano di conseguenza)
- `/<RET>` e `?<RET>` ripetono l'ultima ricerca
- `:s/trova/sostituisci/`
  - cerca 'trova' nella linea corrente e lo sostituisce con 'sostituisci'
- `:%s/trova/sostituisci/cgi`
  - cerca 'trova' in ogni linea del file e lo sostituisce con 'sostituisci'
  - dopo aver chiesto conferma (`c`),
  - anche più volte nella stessa linea (`g`),
  - case-insensitive (`i`)
- `%` è una scorciatoia per `1,$` - in realtà il comando può essere invocato come `:I,Fs/.../.../` per applicarlo alle linee tra `I` e `F`

# Combinazioni e ripetizioni

- La ricerca e gli spostamenti possono essere usati come “terminatore” per alcuni comandi di modifica, ad esempio:
  - **d\$** cancella dalla posizione corrente a fine riga
  - **dG** cancella dalla posizione corrente a fine file
  - **c/ciao<RET>** cancella dalla posizione corrente alla prima occorrenza della stringa ciao e si porta in insert mode
- Digitando il punto **.** si ripete l'ultimo comando impartito
- Facendo precedere un numero N a un comando, il comando viene eseguito N volte consecutivamente
  - Es. **10x** cancella 10 caratteri
- Digitando **u** (undo) si annulla l'ultima azione eseguita



# Copia & incolla (1)

- i comandi di copia utilizzano dei buffer interni a vi; tipicamente si utilizza il buffer standard, ma e' possibile specificarne altri:
  - **yy** copia la linea corrente nel buffer
  - **"a yy** copia la linea corrente nel buffer "a"
- Il comando **d** esegue il cut anziché il copy
- Il paste (incolla) si ottiene con i comandi:
  - **p** incolla dopo la linea corrente
  - **P** incolla prima della linea corrente

# Copia & incolla (2)

- per copiare blocchi di linee si può procedere in diversi modi
  - Usando marcatori
    - ci si posiziona sulla prima linea del blocco
    - **ma** marca la posizione con il simbolo "a"
    - ci si posiziona sull'ultima linea del blocco
    - **y'a** copia nel buffer tutto il testo dalla posizione marcata "a" in precedenza fino alla posizione corrente
  - Usando le ripetizioni
    - ci si posiziona sulla prima linea del blocco
    - per copiare 10 righe si digita **10yy**
  - Usando la ricerca
    - ci si posiziona sulla prima linea del blocco
    - per copiare fino alla parola 'basta' (esclusa) si digita **y/basta<RET>**

# Macro

- Digitando **q<lettera minuscola>** (es: **qa**) inizia la registrazione della macro identificata dalla lettera specificata.
- Tutte le azioni compiute saranno registrate, finché non si preme nuovamente **q** per terminare la registrazione.
- Digitando **@<lettera minuscola>** viene riprodotta la macro.
  - (naturalmente si può riprodurre N volte digitando **N@<lettera>**)